Symbian OS: Overview To Networking

Version 1.0
June 16, 2005
Legal Notice

Copyright © 2005 Nokia Corporation. All rights reserved.

Nokia and Nokia Connecting People are registered trademarks of Nokia Corporation. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.
Contents

1. Introduction ...................................................................................................................................... 7
   1.1 Overview ......................................................................................................................................... 7
   1.2 Scope ............................................................................................................................................... 8

2. Basic Infrastructure ......................................................................................................................... 9
   2.1 Physical Device ............................................................................................................................... 9
   2.2 Device Driver Layer ....................................................................................................................... 9
   2.3 Protocol Implementation Layer ..................................................................................................... 10
       2.3.1 CSY modules .......................................................................................................................... 10
       2.3.2 TSY modules .......................................................................................................................... 10
       2.3.3 PRT modules .......................................................................................................................... 10
       2.3.4 MTM modules .......................................................................................................................... 10
   2.4 Application Layer ......................................................................................................................... 10

3. Symbian OS Client-Server Model .................................................................................................. 11
   3.1 Client - Server Communication ................................................................................................... 11

4. Communications Database ............................................................................................................ 13

5. Serial Communications .................................................................................................................. 14
   5.1 Serial Communications Server Client API .................................................................................... 14
       5.1.1 Server session .......................................................................................................................... 14
       5.1.2 Serial port ............................................................................................................................... 14
       5.1.3 Serial port configuration block ............................................................................................... 14
       5.1.4 Serial port capabilities block ................................................................................................. 15
   5.2 Serial Protocol Module API (CSY) ............................................................................................... 15
       5.2.1 Serial service provider .......................................................................................................... 15
       5.2.2 Serial protocol factory .......................................................................................................... 15
       5.2.3 Serial protocol information ................................................................................................... 15
   5.3 Serial Communication Emulations ............................................................................................... 15

6. Sockets ............................................................................................................................................. 16
   6.1 Socket Server ............................................................................................................................... 16
   6.2 Protocol Modules ........................................................................................................................ 16
       6.2.1 TCP/IP ...................................................................................................................................... 17
       6.2.2 Host resolution services ........................................................................................................ 17
           6.2.2.1 Domain Name Service (DNS) ........................................................................................ 17

7. Telephony ........................................................................................................................................ 19
   7.1 TSY Modules .................................................................................................................................. 19
   7.2 ETel Core API ............................................................................................................................... 20
       7.2.1 Root session ............................................................................................................................ 20
7.2.2 Phone ..........................................................................................................................20
7.2.3 Line ..............................................................................................................................20
7.2.4 Call ..................................................................................................................................20
7.2.5 ETel extension APIs ......................................................................................................20
  7.2.5.1 ETel Multimode API .............................................................................................20
  7.2.5.2 Other extension APIs ............................................................................................21
7.3 Third Party Telephony API ...............................................................................................21
8. Telephony and Networking ...............................................................................................22
  8.1 PDP Context ....................................................................................................................22
  8.2 Multihoming ....................................................................................................................22
  8.3 Quality of Service (QoS) ...............................................................................................23
  8.4 Connection Management API .......................................................................................23
9. Personal Area Networking ..................................................................................................24
  9.1 Bluetooth ........................................................................................................................24
    9.1.1 Bluetooth protocol stack .........................................................................................24
    9.1.2 Bluetooth Stack Settings ........................................................................................25
    9.1.3 Bluetooth APIs ..........................................................................................................26
      9.1.3.1 Bluetooth Service Discovery Agent API ............................................................27
      9.1.3.2 Bluetooth Service Discovery Database API .........................................................27
      9.1.3.3 Bluetooth Security Manager API ........................................................................27
      9.1.3.4 Bluetooth Device Selection UI API ...................................................................27
      9.1.3.5 Bluetooth Sockets API .......................................................................................27
      9.1.3.6 Bluetooth Serial API (RFCOMM) ........................................................................28
  9.2 IrDA ..................................................................................................................................29
    9.2.1 IrDA protocol stack ....................................................................................................29
      9.2.1.1 IrOBEX ...............................................................................................................30
      9.2.1.2 IrTRANP ............................................................................................................30
      9.2.1.3 IrLAN ................................................................................................................30
    9.2.2 IrDA APIs ..................................................................................................................30
      9.2.2.1 IrDA Sockets API ..............................................................................................30
      9.2.2.2 IrDA Serial API (IrCOMM) ...............................................................................31
  9.3 USB ..................................................................................................................................31
10. HTTP and HTML ...............................................................................................................34
  10.1 HTTP Client API .............................................................................................................34
    10.1.1 Session ......................................................................................................................34
    10.1.2 Transaction ...............................................................................................................34
    10.1.3 Headers ....................................................................................................................34
    10.1.4 Data supplier ............................................................................................................35
10.1.5 Filter ........................................................................................................................................35
10.2 HTTP Server API ........................................................................................................................35
10.3 HTML ........................................................................................................................................35
10.4 Web Browsers ..............................................................................................................................35
11. SIP .................................................................................................................................................36
12. Summary .........................................................................................................................................37
13. Terms and Abbreviations ..............................................................................................................38
14. Further Reading .............................................................................................................................40
15. References .....................................................................................................................................41
16. Evaluate This Document ..............................................................................................................42
## Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 16, 2005</td>
<td>Version 1.0</td>
<td>Initial document release</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Overview

This document provides an introduction to Symbian OS networking and communications capabilities.

The main issues are:

- Chapter 2: Basic Infrastructure
- Chapter 3: Symbian OS Client-Server Model
- Chapter 4: Communications Database
- Chapter 5: Serial Communications
- Chapter 6: Sockets
- Chapter 7: Telephony
- Chapter 8: Telephony and Networking
- Chapter 9: Personal Area Networking containing Bluetooth, IrDA, and USB.
- Chapter 10: HTTP and HTML

The following networking and communications capabilities are not described in this document:

- Messaging
- Synchronization
- Security
- WAP
- Java™

Where applicable, the document provides direct links to the Symbian C++ API reference Web site.

There are additional advanced examples available at http://www.forum.nokia.com/ that complement this document and illustrate and provide more detailed information on how to use the security options provided by the platform in practice. The examples include:

- Series 60 Developer Platform 2.0: ECom Plug-in Architecture (With Example)
- Series 60 Developer Platform 2.0: HTTP Client Example
- Series 60 Developer Platform 2.0: IAP Connect Example
- Series 60 Developer Platform 1.0/2.0: Bluetooth Point-to-Multipoint Example
1.2 Scope

This document is intended for project managers, software architects, and developers to get an overview and understanding of what kind of security options and methods are available when working in Symbian OS projects. The essential idioms of networking and communications technology should be understood.
2. **Basic Infrastructure**

The Symbian OS communication infrastructure can be divided into four layers. Figure 1 presents the basic communication layers.

![Diagram of communication layers]

**Figure 1: Basic communication layers**

### 2.1 Physical Device

Symbian OS does not directly control hardware. Hardware is seen by Symbian OS as an abstract unit and it is accessed through this layered API. Using hardware as an abstract unit ensures that Symbian OS can and will handle the wide variety of different devices now and in the future.

### 2.2 Device Driver Layer

The device driver layer contains two sublayers: The *physical device driver* (PDD) layer and the *logical device driver* (LDD) layer. The responsibility of the physical device driver is to communicate directly with the hardware using a specific hardware port whereas the responsibilities of the logical device driver include:
In some cases the same LDD can be used by different device types (for example, infrared port and RS-232 port) but the PDD is typically device-dependent.

Device drivers are developed by the phone manufacturer. Symbian OS provides device drivers for reference hardware only.

Symbian OS permits the run-time loading of LDD and PDD.

2.3 Protocol Implementation Layer

The protocol implementation layer contains several sublayers that communicate with each other in various ways. In Figure 1 some sub-layers are intentionally left blank, meaning that certain types of communication lack some module relationships. For example, the Bluetooth PRT module communicates directly with the device drivers. Protocol implementations are typically developed by device manufactures and third parties in addition to those already supported by Symbian OS.

2.3.1 CSY modules

CSY modules are serial communications server plug-in modules that can be loaded by the serial communications server at run time. CSY modules are used to implement software serial port services. CSYs are described in Chapter 5, “Serial Communications.”

2.3.2 TSY modules

TSY modules are telephony server plug-in modules that can be loaded by the telephony server at run time. TSY modules are used to implement telephony services. TSY modules are described in Chapter 7, “Telephony.”

2.3.3 PRT modules

PRT modules are socket server plug-in modules that can be loaded by the socket server at run time. PRT modules are used to implement protocol services. PRT modules are described in Section 6.2, “Protocol Modules.”

2.3.4 MTM modules

MTM modules are message type modules. MTM modules are used to handle messages of all types. Symbian OS messaging is not in the scope of this document.

2.4 Application Layer

The device manufacturer supplies the device with basic communication applications (for example, telephony) and the additional applications are developed by third parties.
3. **Symbian OS Client-Server Model**

Symbian OS uses the client-server model extensively as an internal pattern. Servers are used to control access to shared resources throughout Symbian OS. Applications written for Symbian OS are usually clients that use resources through servers.

Advantages of the client-server model are:

- Server resources can be shared by multiple applications.
- Servers provide functional abstraction, which means that clients can use the same API without needing to know the hardware (device) or software protocol providing the required service.
- Less possibilities for errors because clients and servers are isolated from each other.

3.1 **Client - Server Communication**

Client - server communication can be divided into five phases:

- **Initialization**: In the first phase the physical/logical device drivers and communication components are loaded as needed. Loaded communication components may vary depending on the server type. More detailed explanations about the loaded communication components are given in the description of the server in question. In addition, the connection is established with the server. Sometimes the server must be started before the connection is made if it is not already running.

- **Opening**: In the opening phase the OS sets up the internal housekeeping for the device, checks access issues, and sets up protection mechanisms.

- **Configuring**: A communication device must be configured before the data can be exchanged between the client and the server. Configuring can be accomplished by telling the device to use the settings in the communications database (see Chapter 4, “Communications Database”) or configuration parameters can be set using function calls.

- **Data exchange**: In this phase the actual data is exchanged between the client and the server. Data is exchanged using a specified protocol. More detailed explanations about supported communication protocols are given in later chapters in the description of the server in question.

- **Closing**: In the final phase the communication device can be closed to free up the resources. The connection to the server is also closed.

The main servers (see Figure 2) in Symbian OS communications are:

- Communications Database Server (see Chapter 4, “Communications Database”).
- Serial Communications Server (see Chapter 5, “Serial Communications”).
- Telephony Server (see Chapter 7, “Telephony”).
- Socket Server (see Chapter 6, “Sockets”).
- Host Resolver Server (see Section 6.2.2, “Host resolution services”).

Version 1.0  |  June 16, 2005
Figure 2: Symbian OS communication components and their main relationships
4. Communications Database

The communications database (CommDb) is a Symbian OS database managed by the Symbian OS DBMS system and it provides a system-wide storage for communications-related settings. The CommDb is based on client/server architecture and it holds information about:

- Internet Access Providers (IAPs)
- Internet Service Providers (ISPs)
- GPRS
- Modems
- Locations
- Charge cards
- Proxies
- WAP settings.

The communications database is used by applications and other communications sub-systems (such as Networking, Telephony, and WAP) that require component configuration information. The content of the database can be read and updated by the Control Panel in the Symbian OS user interface (user’s point of view) or by using the Symbian OS CommDb API (application point of view). The CommDb API allows multiple clients to access the tables within the communications database concurrently. The locking mechanism of the underlying DBMS allows concurrent access by multiple clients while safeguarding the integrity of the database for changes.
5. **Serial Communications**

This chapter gives an overview of the serial communications facilities that Symbian OS provides.

The heart of the Symbian serial communication is the serial communications server (C32) that is based on a Symbian OS client/server framework. The serial communications server provides a framework for serial communications services. Clients (and the Symbian OS sockets server and telephony server) use R classes that send requests to the serial communications server. The server in turn passes requests to an appropriate plug-in module that handles the particular communication protocol. These plug-in modules, known as CSY modules or serial protocol modules (see Sections 2.3.1, “CSY modules” and 5.2, “Serial Protocol Module API (CSY)”), are loaded by the serial communications server, and are not directly accessed by client applications. Serial protocol module implementations typically call low-level device driver interfaces (LDD and PDD) to access their appropriate communications hardware. A Symbian OS phone may include at least the following CSY modules for handling serial communications:

- RS232 (ECUART.CSY)
- Infrared (IRCOMM.CSY)
- Bluetooth (BTCOMM.CSY)

The Serial Protocol Module API allows developing new CSY modules.

5.1 **Serial Communications Server Client API**

The client API has four key concepts: server session, serial port, serial port configuration block, and serial port capabilities block.

5.1.1 **Server session**

The server session allows the client to:

- Discover the serial protocols available on the device, and select the appropriate one.
- Discover the ports available on the device.

The server session interface is provided by RCommServ.

5.1.2 **Serial port**

The serial port allows the client to access a particular hardware serial port. The interface is used to read, write, configure, set break conditions, and get port state information.

The serial port interface is provided by RComm.

5.1.3 **Serial port configuration block**

The serial port configuration block specifies the operation of a serial port. This includes settings such as the data rate, parity type, and handshaking control. A client sets up a serial port before use by providing a configuration block.
The serial port configuration block interface is provided by TCommConfigV01.

### 5.1.4 Serial port capabilities block

The serial port capabilities block specifies the capabilities of a serial port. This indicates the possible configuration settings for a port. A client can get the capabilities block for a serial port before configuring it, to ensure that the desired configuration is possible.

The serial port capabilities block interface is provided by TCommCapsV01.

### 5.2 Serial Protocol Module API (CSY)

The **Serial Protocol Module API** (cs_port.h) has three (3) concepts: serial service provider, serial protocol factory, and serial protocol information.

#### 5.2.1 Serial service provider

The serial service provider defines an interface that encapsulates a port, with methods largely corresponding to the port interface provided to clients by the Serial Communications Server Client Side API. A serial protocol module implements the interface to access the hardware ports for its protocol.

The serial service provider interface is provided by CPort.

#### 5.2.2 Serial protocol factory

The serial protocol factory defines an interface that the Serial Communications server calls to create serial service providers. A serial protocol module implements the interface to provide a factory for its own serial service provider objects.

The serial protocol factory interface is provided by CSerial.

#### 5.2.3 Serial protocol information

Serial protocol information specifies information about the ports, such as name and number, accessible through the protocol. A serial protocol module provides suitably configured protocol information to the server.

The serial protocol information interface is provided by TSerialInfo.

### 5.3 Serial Communication Emulations

IrDA serial port emulation is supported by the IrDA serial communications server module called IRCOMM.CSY. Serial communication via IrDA is described in Section 9.2.2.2, “IrDA Serial API (IrCOMM).”

Bluetooth serial port emulation is supported by the Bluetooth communications server module BTCOMM.CSY, which provides a number of thin virtual serial ports for different legacy services running over RFCOMM socket functionality. Serial communications via Bluetooth is described in Section 9.1.3.6, “Bluetooth Serial API (RFCOMM).”

Serial communications via USB is described in Section 9.3, “USB.”
6. Sockets

This chapter gives an introduction to the Symbian OS sockets API.

The sockets API is the heart of the networking capability of Symbian OS. The sockets API is provided by the sockets server (ESOCK) and it is based on a Symbian OS client/server framework. Clients use classes that send requests to the sockets server. The server in turn passes requests to an appropriate plug-in module that handles the particular communication protocol. These plug-in modules, known as PRT modules or protocol modules (see Section 6.2, “Protocol Modules”), are loaded by the socket server and are not directly accessed by client applications. Protocol module implementations typically call low-level device driver interfaces (LDD and PDD) to access their appropriate communications hardware. A Symbian OS phone may include at least the following PRT modules for handling protocols:

- TCP/IP (TCPIP.PRT)
- Infrared (IRDA.PRT)
- Bluetooth (BT.PRT)

The Protocol Module API allows new PRT modules to be developed and to provide sockets services over different transports.

6.1 Socket Server

The Symbian OS socket server is based on the client/server architecture (see Chapter 3, “Symbian OS Client-Server Model”) and it provides a BSD-like C sockets API to clients (STDLIB). Symbian OS provides also C++ socket APIs that are not identical to the traditional BSD C API. The four most important classes are:

- **RSocketServ**: This class establishes and obtains necessary resources for the connection to the sockets server. In client-server terminology, this class represents the application’s session with the sockets server. All other client interface classes require a session to the server to be opened via an instance of this class.
- **RSocket**: This class represents a socket. A typical application will have several instances of RSocket operational at any time.
- **RHostResolver**: This class offers an interface for host name resolution services ()
- **RNetDatabase**: This class offers an interface for network database access.

RSocket, RHostResolver, and RNetDatabase all represent subsessions within the context of a given application’s session to the sockets server — that is, an RSocketServ instance.

Socket interfaces are declared in es_sock.h and ss_std.h.

6.2 Protocol Modules

The socket server provides application programmers a multiple transport extensible networking API. To support this, the socket server makes use of dynamically loadable
plug-in libraries known as protocol modules (PRT modules). Protocol modules are dynamic-link libraries (DLLs) with a `.PRT` extension.

A single protocol module can contain one or more related protocols (a protocol family). Examples of protocol families are TCP/IP (see Section 6.2.1, “TCP/IP”) and IrDA. Protocol families must be loaded and unloaded dynamically, they must be able to identify themselves and their capabilities, and (some at least) must provide socket services. Protocol families may also provide service registration and name space resolution services. Not all protocols need to provide socket access — for instance; IrLAP does not provide socket access.

Via the Socket server protocols API (es_prot.h, es_sock.h) developers can implement new protocol modules to provide socket services over different transports.

### 6.2.1 TCP/IP

TCP/IP communication services are provided to client applications through the TCP/IP protocol module (TCPIP.PRT). This module provides the following four fundamental transport and network layer protocols:

- Internet Protocol (IPv4/v6 dual stack).
- Internet Control Message Protocol (ICMP).
- User Datagram Protocol (UDP).
- Transmission Control Protocol (TCP).

The TCPIP.PRT protocol module is not accessed directly by clients, but used through the Sockets Client API.

The module also provides a DNS (see Section 6.2.2.1, “Domain Name Service (DNS)”) service that can be accessed through the Sockets Client API’s standard host name resolution interface (see Section 6.2.2, “Host resolution services”).

TCP/IP services can be used over many network interface types. To enable access to TCP/IP networks through a dial-up serial connection, the RGenericAgent plug-in is used to establish a connection before setting up a PPP connection.

### 6.2.2 Host resolution services

As part of its client side API, the sockets server offers the class RHostResolver, which provides a generic interface to protocol-specific host resolution services. In case of the TCP/IP protocol suite, the RHostResolver class acts as an interface to the Domain Name Service, or DNS (see Section 6.2.2.1, “Domain Name Service (DNS)

Each different protocol that offers a host resolution service will provide the implementation of that service as part of the relevant protocol module. The RHostResolver client interface is used across all protocols that provide such a service.

#### 6.2.2.1 Domain Name Service (DNS)

Domain Name Service (DNS) is the host resolution service provided by TCP/IP.

A typical DNS query involves three steps:
1. A client application, running on one networked device, sends a resolution request to another networked machine – a DNS server.

2. The DNS server takes the supplied address and looks it up in a large table of addresses to translate it to a different address format.

3. The DNS then returns the alternative address to the client.

DNS can be used to translate the textual address format, such as www.symbian.com, to the numeric address format, for example 212.134.93.203, or the numeric address, such as 204.71.202.160, to the user-friendly textual address, for example www.yahoo.com.

Internet service providers (ISP) provide a number of networked DNS servers (usually more than one) for their customers to use. The important point is that the DNS server does the actual address translation for the client device and therefore the numeric address of the ISP’s DNS server is a key part of the configuration of an Internet connection established over TCP/IP.
7. **Telephony**

This chapter gives an introduction to the Symbian OS telephony API.

The telephony API uses the Symbian client/server framework called **ETel telephony server** and it provides a standard API which enables its clients to initiate, control, and terminate data, fax, and voice calls using the same methods for any hardware. The API provides R classes that send requests to the telephony server. The telephony server in turn passes requests to an appropriate plug-in module that handles the physical device. These plug-in modules, known as **TSY modules or telephony extension modules** (see Section 7.1, “TSY Modules”), are loaded and unloaded by the telephony server at the request of clients. Symbian OS provides, as a standard, generic modules for GSM mobile devices and also for terrestrial modems using AT command sets.

![ETel architecture overview](image)

### 7.1 TSY Modules

ETel telephony extension modules — TSYs — translate between client side API function calls and the instructions understood by a particular telephony device, or family of devices.

There is no requirement for a particular TSY to support the complete set of functions defined in the client-side APIs, although typically a TSY will support — as a minimum — all the functions in the core client side API. ETel returns an error if a TSY does not include a called function.

The TSY API allows new TSY modules to be developed.

Symbian OS provides two main APIs for application programmers:

- ETel Core API (removed in Symbian OS v9)
- Third Party Telephony API (fully supported from Symbian OS v9 onwards).
7.2 ETel Core API

The ETel Core API (declared in etel.h and externor.h) has four key concepts: root session, phone, line, and call. Note that this API has been marked as deprecated in Symbian OS v8.0a. See Section 7.3, “Third Party Telephony API,” for information on the replacing API.

7.2.1 Root session

The root session provides access to system telephony information, in particular the available devices and TSYs.

The root session interface is provided by RTelServer.

7.2.2 Phone

The phone abstracts a particular telephony device. The interface allows the client to access the status and capabilities of the device, and to be notified if these change.

The phone interface is provided by RPhone.

7.2.3 Line

A phone can have one or more lines. Similarly to the phone, clients can access the status and capabilities of a line, and can be notified if these change.

The line interface is provided by RLine.

7.2.4 Call

Lines can have zero or more active calls. A call has the functionality to dial a number, to wait for an incoming call, and to hang up. As before, a client can get status and capabilities information, and can be notified of changes.

The call interface is provided by RCall.

7.2.5 ETel extension APIs

7.2.5.1 ETel Multimode API

The ETel Multimode API extends the ETel Core API to offer access to common mobile telephony services for a number of air-interface cellular standards. The supported standards are:

- Global System for Mobile Communications (GSM): The GSM telephony framework provides an abstract telephony interface for GSM voice and data and for landline modems for data as well as phone number resolution and the SIM Application Toolkit.

- General Packet Radio Service (GPRS): The GPRS framework provides an abstract telephony interface for GPRS class B functionality. GPRS Release 97/98, Release 99 (GPRS and UMTS), and Release 4 (UMTS) packet services as well as CDMA/CDMA2000 are the specifications implemented. With class B functionality devices are able to make and receive GSM calls while simultaneously remaining registered with GPRS. If a Packet Data
Protocol context is active, GPRS services are automatically suspended and resumed.

- Enhanced Data Rates for Global Evolution (EDGE): The EDGE framework provides an abstract telephony interface for 3GPP GSM/EDGE.
- Code-Division Multiple Access (CDMA) (IS-95): The CDMA telephony framework provides an abstract telephony interface for CDMA (IS-95) voice and data (circuit- and packet-switched).
- 3GPP2 CDMA2000 1x (Release A): The 3GPP2 cdma2000 1x telephony framework provides an abstract telephony interface for 3GPP2 cdma2000 1x (Release A) voice and data (circuit- and packet-switched).

7.2.5.2 Other extension APIs

The following extension APIs are also available:

- ETel Multimode Packet Data
- **SIM Application Toolkit** (Etelsat.h, satcs.h)
- **Dial** (dial.h)
- **PhoneBook Synchronizer** (phbksync.h)
- **Fax Client** (cfax32.h, etel.h, faxdefn.h, faxset.h, faxstd.h)
- **Fax Header Line** (cfaxio.h)
- **Fax Store** (faxstore.h)

7.3 Third Party Telephony API

The third party telephony API (fully supported from Symbian OS v9 onwards) provides a high-level interface for making and receiving telephony data calls. The API wraps access to the ETel Core API, to which it provides a simpler, if less flexible, interface. The third party telephony API allows the caller to make an outgoing telephone call, or wait for an incoming call. Synchronous and asynchronous functions for both operations are available. The caller passes an RComm (serial port) object to the object, and when the call is established, it uses that object to read and write data over the connection. Besides call functions, the API provides access to IMEI (phone serial number) and IMSI (subscriber ID) features. The third party telephony API is declared in a header file called Etel3rdParty.h.
8. Telephony and Networking

The Symbian OS communications and networking architecture supports the following features:

- Multiple primary and secondary PDP contexts
- Multihoming
- Quality of service (QoS).

8.1 PDP Context

A PDP (packet data protocol) is a network protocol used by external packet data networks that communicate with a GPRS network. IP is an example of a PDP supported by GPRS. The PDP context refers to a set of information (such as a charging ID) that describes a mobile wireless service call or session, which is used by mobile devices in a GPRS network to identify the session.

A PDP context can be thought of as a single connection to a network. PDP contexts can be further broken down into two types – primary and secondary. A primary PDP context is a connection from the mobile phone to the network with one IP address allocated to it. A secondary PDP context is one that shares an IP address with a primary PDP context, but that represents another connection to the network. Each PDP context can have a QoS (see Section 8.3, “Quality of Service (QoS)”) associated with it that helps prioritizing network traffic.

![Figure 4: PDP context](image)

8.2 Multihoming

Symbian OS supports (from Symbian OS v7.0s onwards) multiple concurrent IP addresses (see Section 8.4, “Connection Management API”), an ability described as multihoming. Multihoming allows connecting to multiple GPRS and WCDMA services simultaneously. Symbian OS does not limit multihoming to the telephony side; it also
allows different IP addresses for other services such as Bluetooth wireless technology, PC connectivity, or Ethernet. Only the availability of system resources limits the number of simultaneous connections, and it is possible to dynamically load or unload them without restarting the TCP/IP stack. Multiple IAPs can be active and applications can specify which ones they will use. Support also exists to allow applications to specify Quality of Service parameters on GPRS/UMTS networks supporting Secondary PDP Contexts.

![Figure 5: Accessing multiple services in GPRS and 3G networks](image)

8.3 Quality of Service (QoS)

The Symbian OS Quality of Service API (qoslib.h) allows devices to manage network connections intelligently, ensuring the best possible user experience in any given situation. For example, the device may provide a high bandwidth guaranteed network connection for browsing and a low priority “best effort” network connection for downloading e-mails. This results in a great browsing experience for the user whilst e-mails are downloaded in the background. The Quality of Service API was introduced in Symbian OS v8.0a. In Series 60, it is supported from Series 60 Platform 2 nd Edition, Feature Pack 3 onwards.

8.4 Connection Management API

Handling of connections in a multihomed environment is done using the Connection Management API (ES.SOCK.h), supported from Symbian OS 7.0s onwards. The Connection Management interface is provided by RConnection and it allows applications to utilize multiple concurrent IP addresses (see Section 8.2, “Multihoming”).
9. Personal Area Networking

This chapter provides an overview of the Symbian Personal Area Networking (PAN).

9.1 Bluetooth

This section discusses the Bluetooth communications facilities that Symbian OS provides.

9.1.1 Bluetooth protocol stack

Bluetooth protocol stack layers are presented in Figure 6.

![Bluetooth Protocol Stack Diagram](image)

**Figure 6: Bluetooth protocol stack**

The Bluetooth stack contains the following layers:

- Bluetooth Radio: This layer is the physical layer of the Bluetooth stack. Bluetooth uses radio frequencies in the band allocated at 2.4 GHz.
- Baseband Link Controller. This layer is responsible for encoding and decoding packets, link control signaling (flow control, acknowledgement/retransmission of signals).
- Link Manager Protocol (LMP): This layer is used to establish, secure, and control the links between devices. LMP also controls transmission power and the layer is responsible for monitoring the link between devices and watching for link quality.
- **BT Host Controller (HW):** The Bluetooth Host Controller components provide the lower level of the stack. The Host Controller components are normally implemented in the hardware. Applications do not have direct access to this layer.

- **L2CAP:** The Logical Link Control and Adaptation Protocol layer provides asynchronous connectionless (ACL) data services to the upper layer protocols with protocol multiplexing capability, segmentation and reassembly operation, and group abstractions. L2CAP supports only ACL links and no SCO links.

- **SDP:** The Service Discovery Protocol layer is used to query device services and characteristics of services before the actual connection between two or more Bluetooth devices can be established. SDP is utilized by the Service Discovery Agent described in Section 9.1.3.1, “Bluetooth Service Discovery Agent API.”

- **RFCOMM:** This layer is not a core layer but it is built on top of the core layers. RFCOMM emulates an RS-232 serial line and presents an interface to upper layer protocols, for example, OBEX.

The Symbian OS Bluetooth APIs give applications access to RFCOMM, L2CAP, SDP, OBEX and, to a limited extent, HCI. Note that there are several changes in the Bluetooth architecture in Symbian OS v8.0a, some of which also affect third party APIs.

### 9.1.2 Bluetooth Stack Settings

In Symbian OS v.8.0, a publish and subscribe method (bt_subscribe.h) is provided for setting the stack settings, which were managed via IOCTLs and other APIs in the earlier versions. The Bluetooth stack publishes the values of its various settings in the category `KPropertyUidBluetoothCategory` and applications can request that settings are changed by setting the value of the equivalent property in the `KPropertyUidBluetoothControlCategory`. Therefore a typical pattern for an application wishing to set the device discoverable would be:

- Subscribe to the key of interest in the Bluetooth category.
- Call the `RProperty::Set(...)` method on the matching key in the Bluetooth Control category.
- When the stack has serviced the request, it will re-publish the new value of the key (whether changed or unchanged) on the Bluetooth category, calling the `RunL` of any subscribers.

The available property keys are described in Table 1.

<table>
<thead>
<tr>
<th>Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPropertyKeyBluetoothLocalDeviceAddress</td>
<td>ByteArray</td>
<td>The Local Device Address as a descriptor (BluetoothCategory only).</td>
</tr>
<tr>
<td>KPropertyKeyBluetoothPHYCount</td>
<td>Integer</td>
<td>The number of currently connected Bluetooth Physical Links:</td>
</tr>
</tbody>
</table>
### Table 1: Property keys

<table>
<thead>
<tr>
<th>Property Key</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPropertyKeyBluetooth Connecting</td>
<td>Integer</td>
<td>1 - if actively paging another device (i.e., connecting). 0 - if not paging another device.</td>
</tr>
<tr>
<td>KPropertyKeyBluetooth Scanning</td>
<td>Integer</td>
<td>The current scanning mode.</td>
</tr>
<tr>
<td>KPropertyKeyBluetooth LimitedDiscoverable</td>
<td>Integer</td>
<td>Value from the enum THCIScanEnable: ENoScansEnabled, EInquiryScanOnly, EPageScanOnly, or EInquiryAndPageScan</td>
</tr>
<tr>
<td>KPropertyKeyBluetooth DeviceClass</td>
<td>Integer</td>
<td>The full integer value of the current Class of the device.</td>
</tr>
<tr>
<td>KPropertyKeyBluetooth RegistryTableChange</td>
<td>Integer</td>
<td>The index of the table which was changed.</td>
</tr>
</tbody>
</table>

#### 9.1.3 Bluetooth APIs

Figure 7 shows how the different Bluetooth APIs relate. The Bluetooth Sockets API is the fundamental API upon which the other APIs rely to communicate with other devices.

![Bluetooth API relationships](image.png)

**Figure 7: Bluetooth API relationships**
9.1.3.1 Bluetooth Service Discovery Agent API

The **Bluetooth Service Discovery Agent API** (btsdp.h) is used to discover which services are available on a remote device (see Section 9.1.3.2, "Bluetooth Service Discovery Database API"), as well as the attributes of those services. Services are discovered by using the SDP protocol although the Service Discovery Agent (CSdpAgent) encapsulates the use of SDP. The BT.PRT socket server plug-in generates a distinct Service Discovery Protocol (SDP) server thread that handles SDP queries and responses.

9.1.3.2 Bluetooth Service Discovery Database API

The **Bluetooth Service Discovery Database** (btsdp.h) is a local server-managed database (see Chapter 3, "Symbian OS Client-Server Model") that contains Bluetooth-related records and attributes, so that remote devices may discover its presence and determine if it can be used. Creating and registering a service is an essential part of a Bluetooth application. See [How to register a service in the database](www.symbian.com) for more information about service creation and registration.

9.1.3.3 Bluetooth Security Manager API

The **Bluetooth Security Manager API** (btmanclient.h) enables services to set appropriate security requirements, which incoming connections must meet.

Changes in Symbian OS v8.0a

The methods `RBTSecuritySettings::RegisterService` and `RBTSecuritySettings::UnregisterService` used for configuring security for incoming connections are now replaced by `TBTSockAddr::SetSecurity`, which sets the security when the listening socket is opened. This is also true for outgoing connection sockets, which replace IOCTLs `KHCIEncryptIoctl` and `KHCIAuthRequestIoctl`. They now return `KerrNotSupported`.

Service security overrides are set on the particular listening socket to which they apply by calling `RSocket::SetOption [level: KSolBtL2CAP, name KBTSecurityDeviceOverride]` on the socket in question. This replaces the methods `CBTDevice::SetServiceSecurityL`, `DeleteServiceSecurityL`, and `SetIndividualSecurityArrayL`. The class `TBTDeviceSecurityPerService` has been replaced by `TBTServiceSecurityPerDevice`.

9.1.3.4 Bluetooth Device Selection UI API

The **Bluetooth Device Selection UI API** (btdevice.h, btextnotifiers.h, btmanclient.h, bttypes.h) provides an interface by which a dialog asking users for device selection information can be called.

9.1.3.5 Bluetooth Sockets API

The **Bluetooth Sockets API** (bttypes.h, bt_sock.h) is exposed through the standard sockets interface (ESOCK) using the BT.PRT plug-in component. The BT.PRT provides support for both the Logical Link Control and Adaptation Protocol (L2CAP) and the RFCOMM protocol layers through a TCP/IP-like sockets interface.
**Changes in Symbian OS v8.0a**

The `RSocket::SetLocalPort(port number)` API (defined in `es_sock.h`), which is used to specify the channel that should be listened to, has been deprecated.

Applications should instead call the `RSocket::Bind(sockAddr)` method (defined in `es_sock.h`) specifying a `sockAddr` with the port set to either `KRFCOMMPassiveAutoBind` or `KL2CAPPassiveAutoBind` as appropriate. These constants are defined in `bt_sock.h`. `RSocket::LocalPort` and should be used to find out the port.

**SCO connections**

The way of creating and manipulating SCO connections via IOCTLS on connected RFCOMM sockets has been deprecated. A generic API for dealing with SCO connections has been introduced (defined in `bt_sock.h`).

9.1.3.6 **Bluetooth Serial API (RFCOMM)**

The Bluetooth Serial API is exposed through the communications server plug-in module `BTCOMM.CSY`. The Bluetooth `BTCOMM.CSY` is a polymorphic DLL loaded by the communications server to provide serial port emulation. The Bluetooth communications server module supports outgoing connections only. This is used for legacy PC datasync support through Symbian Connect and for the Serial Port Profile. Incoming serial connections must be implemented using RFCOMM sockets.
9.2 IrDA

This chapter provides an overview of the IrDA communications facilities that Symbian OS provides.

9.2.1 IrDA protocol stack

The Symbian OS infrared IrDA stack is contained in a socket server protocol module (IRDA.PRT). IrDA stack layers are presented in Figure 9.

Figure 9: IrDA protocol stack (socket server plug-in module)

The IrDA protocol stack implements the following IrDA layers:
- IrLAP v1.1
- IrLMP v1.1
- IrTinyTP v1.1.

The following features are supported:
- Slow infrared (SIR) supporting throughputs of 9.6 Kbps to 115.2 Kbps.
- Fast infrared (FIR) supporting throughputs of 0.576 Mbps to 4 Mbps.
- IrOBEX v1.2 (IrDA object exchange).
- IrTRANP v1.0 digital camera picture infrared transfer protocol.
- IrCOMM v1.0 emulation of the serial and parallel ports, supports fax/modem functionality and is implemented in a serial communications server module.

Functionality of the infrared is provided by the IrDA Sockets API (see Section 9.2.2.1, “IrDA Sockets API”) and the IrDA Serial API (see Section 9.2.2.2, “IrDA Serial API (IrCOMM)”).

9.2.1.1 IrOBEX

Infrared Object Exchange Protocol is used to exchange data objects between two devices.

9.2.1.2 IrTRANP

The Infrared Transfer Picture Protocol API (transp.h) provides transfer of pictures between digital cameras and Symbian OS phones over infrared.

9.2.1.3 IrLAN

Infrared Local Area Network Protocol is used to connect a device to the local area network. The IR sender device usually connects to the IR-enabled receiver device that is directly connected to a network. The receiver device becomes a gateway to the network. IrLAN can also be used to connect two IR devices and enable communicating as if the devices were on a larger network. IrLAN allows devices to communicate using network protocols.

9.2.2 IrDA APIs

9.2.2.1 IrDA Sockets API

The Infrared Sockets API (ir_sock.h) allows applications to use infrared communications through the generic sockets interface of Symbian OS. The sockets server defines a generic interface for all socket-type communication services, and defines a plug-in architecture for implementing particular providers. The IRDA.PRT module is such a plug-in. A client accesses its services through the generic sockets client API, specifying protocol-specific behavior through infrared-specific utility types and constants.
9.2.2.2 **IrDA Serial API (IrCOMM)**

The IrDA CSY API provides emulation of serial port communications over infrared using the IRCOMM.CSY plug-in module. Its main use is to make infrared available to legacy applications designed to use serial ports. Similar to the IrDA sockets API stated above, the Symbian OS serial communications server defines a generic interface for all serial type communication services, and defines a plug-in architecture for implementing particular providers. The IRCOMM.CSY module is such a plug-in. A client accesses its services through the generic serial communications server client-side API. The plug-in provides a layer over the IrDA sockets API.

The use of IRCOMM is not recommended because it hides some of the useful features of the lower protocols and the traffic runs always at 9600 bps.

![Diagram of the Symbian OS implementation of the IrDA protocol suite](image)

**Figure 10: The Symbian OS implementation of the IrDA protocol suite**

9.3 **USB**

Symbian OS provides (from Symbian OS v7 onwards) access to USB client hardware through its **USB Client API**. The implementation needs to be customized for different licensee products.
The USB client hardware interface contains multiple endpoints, where this number can be anything between one and thirty-one. A device that contains a large number of endpoints can support a number of USB functions simultaneously.

The Symbian OS design allows endpoints to be grouped into one or more USB interfaces, where each interface will, typically, be used for a different USB function or subfunction.

The API has three main elements:

- The kernel-side USB Client Driver, also known as the logical channel, LDD, or simply the channel.
- The kernel-side USB Client Controller, which manages access to the USB client device for all USB client drivers.
- The user-side handle to the USB client driver.

The API supports:

- Multiple USB interfaces, that is, it supports licensee products that implement multiple USB interfaces simultaneously.
- Alternate USB interfaces.
- A single USB configuration only; the API does not provide a mechanism for specifying more than one USB configuration.

Figure 11 illustrates the USB client architecture. It shows the components involved if a licensee product were to be configured as, for example, a dual mass storage / audio peripheral.
USB client architecture

Figure 11: USB client architecture
10. HTTP and HTML

This chapter gives an introduction to the Symbian OS HTTP and HTML support.

10.1 HTTP Client API

Symbian OS provides an HTTP 1.1-compliant client API for communicating with HTTP servers on the Internet. The interface architecture provides a generalized mechanism for HTTP-like protocols that operate over various transports. Using a single API, a client can choose an HTTP protocol, encoding, and transport.

The internal implementation of the HTTP Client API is asynchronous and it communicates with client implementations through a callback mechanism. No custom Active Objects or Sockets API implementations are needed, because the internal implementation of the HTTP API hides (encapsulates) them.

The HTTP client API has five key concepts: session, transaction, headers, data supplier, and filter.

10.1.1 Session

Session encapsulates a client's HTTP activity over the duration of the client’s execution. It defines properties for HTTP transactions (see Section 10.1.2, “Transaction”), such as protocol, encoding, and used transport. Also, a set of filters (see Section 10.1.5, “Filter”) can be associated with the session to provide additional automatic behavior, such as basic HTTP authentication.

The session interface is provided by RHTTPSession.

From Symbian OS v7.0s onwards, the HTTP framework supports multi-homed (see Section 8.2, “Multihoming”) devices. HTTP clients can specify a connection preference through the session property HTTP::EHttpResponseConnection.

10.1.2 Transaction

Transaction represents an interaction between the HTTP client and server and consists of an HTTP header and body. A transaction can be either an HTTP request or a response as defined in the HTTP protocol. Transactions execute asynchronously within the client implementation, and a callback mechanism is used to pass transaction events, such as a notification when the header of a response is received.

The transaction interface is provided by RHTTPTransaction.

10.1.3 Headers

Headers represents the headers of an HTTP request or response, such as content length and type.

The headers interface is provided by RHTTPHeaders.
10.1.4 Data supplier

Data suppliers encapsulate the body of a transaction for either request or response.

The abstract data supplier interface is provided by MHTTPDataSupplier.

10.1.5 Filter

Filters are ECom add-on modules that provide additional behaviors to a session beyond the simple request-response transaction, for example, authentication that would otherwise require multiple requests and responses to be transferred between the client and the server in the correct order and format.

The abstract filter interface is provided by MHTTPFilter.

10.2 HTTP Server API

Symbian OS does not contain direct built-in support for HTTP servers. HTTP servers must be implemented using the socket API.

See the SmallServ example application for a simple HTTP server implementation for Symbian OS.

10.3 HTML

Symbian OS does not contain built-in support for HTML viewing/rendering. The viewing of HTML content is the responsibility of an application, and there is no built-in support for HTML viewers that can be embedded in application and GUI components.

10.4 Web Browsers

Symbian OS does not have its own Web browser and browser development is left to licensees or specialists, such as Opera (see http://www.opera.com/). The Browser control and Download manager APIs (introduced in Series 60 Platform 2nd Edition, Feature Pack 3) are examples of browser-related interfaces developed by a licensee.
11. SIP

Session Initiation Protocol (SIP) is a de facto application-layer-signaling protocol for creating, modifying, and terminating session(s) with participant(s). SIP extensions are used, for example, for messaging, subscribing to events, receiving event notifications, and publishing presence information. SIP has been adopted by the 3rd Generation Partnership Project (3GPP) and it is used in IMS (IP Multimedia Subsystem) Release 5. SIP can run on top of several different transport protocols.

The SIP protocol differs from previous protocols in being an application layer protocol. Previous protocols have typically been transport layer protocols. SIP is not a transport protocol (although it can be used to carry content). Instead, it uses transport protocols such as UDP, TCP, SCTP, or even SMS.

Integrated SIP protocol support is included in Series 60 Platform 3rd Edition based on Symbian OS v9.1. Earlier versions have SIP implementations from various vendors. For example, a SIP implementation with a SIP Client API has been developed for Series 60 Platform 2nd Edition, Feature Pack 1. The API provides access to the basic services of a SIP stack and is appropriate for applications that intend to use SIP for managing multimedia sessions, subscribing to events, and sending stand-alone SIP messages.
12. Summary

This document gave an introduction to Symbian OS networking and communications capabilities. Communication technology is a rapidly changing and evolving area of Symbian OS. More information about which APIs in the Symbian Developer Library are available on any particular Symbian OS phone/version can be obtained from http://www.symbian.com/developer/techlib/papers/SymbOS_cat/SymbianOS_cat.html
### 13. Terms and Abbreviations

<table>
<thead>
<tr>
<th>Term or abbreviation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3GPP</td>
<td>3rd Generation Partnership Program</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>EDGE</td>
<td>Enhanced Data Rates for Global Evolution</td>
</tr>
<tr>
<td>ESOCK</td>
<td>Symbian sockets server</td>
</tr>
<tr>
<td>GPRS</td>
<td>General Packet Radio Service</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile Communications</td>
</tr>
<tr>
<td>HCI</td>
<td>(Bluetooth) Host Controller Interface</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>ICMP</td>
<td>Internet Control Message Protocol</td>
</tr>
<tr>
<td>IMS</td>
<td>(3GPP) IP Multimedia Subsystem</td>
</tr>
<tr>
<td>IPv4</td>
<td>Internet Protocol Version 4</td>
</tr>
<tr>
<td>IPv6</td>
<td>Internet Protocol Version 6</td>
</tr>
<tr>
<td>IrCOMM</td>
<td>Infrared serial communications protocol</td>
</tr>
<tr>
<td>IrDA</td>
<td>Infrared Data Association</td>
</tr>
<tr>
<td>IrLAN</td>
<td>Infrared Local Area Network Protocol</td>
</tr>
<tr>
<td>IrOBEX</td>
<td>Infrared Object Exchange Protocol</td>
</tr>
<tr>
<td>IrTRANP</td>
<td>Infrared Transfer Picture Protocol</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>LC</td>
<td>(Bluetooth) Baseband Link Controller</td>
</tr>
<tr>
<td>L2CAP</td>
<td>(Bluetooth) Logical Link and Control Adaptation Protocol</td>
</tr>
<tr>
<td>LMP</td>
<td>(Bluetooth) Link Manager Protocol</td>
</tr>
<tr>
<td>LDD</td>
<td>Logical Device Driver</td>
</tr>
<tr>
<td>OBEX</td>
<td>Object Exchange Protocol</td>
</tr>
<tr>
<td>PAN</td>
<td>Personal Area Networking</td>
</tr>
<tr>
<td>PDD</td>
<td>Physical Device Driver</td>
</tr>
<tr>
<td>Protocol</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>PDP</td>
<td>Packet Data Protocol</td>
</tr>
<tr>
<td>RFCOMM</td>
<td>(Bluetooth) serial port emulation protocol</td>
</tr>
<tr>
<td>SCTP</td>
<td>Stream Control Transmission Protocol</td>
</tr>
<tr>
<td>SDP</td>
<td>Service Discovery Protocol</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>WAP</td>
<td>Wireless Application Protocol</td>
</tr>
<tr>
<td>WCDMA</td>
<td>Wideband CDMA (Code-Division Multiple Access)</td>
</tr>
</tbody>
</table>
14. Further Reading

1. Forum Nokia provides numerous documents and examples:
   http://www.forum.nokia.com/

2. Technical papers on the Symbian Web site cover communications and networking.
15. References

Series 60 Developer Platform 2.0: ECom Plug-in Architecture (With Example)
Series 60 Developer Platform 2.0: HTTP Client Example
Series 60 Developer Platform 2.0: IAP Connect Example
Series 60 Developer Platform 1.0/2.0: Bluetooth Point-to-Multipoint Example
Available at Forum Nokia, http://www.forum.nokia.com/
Symbian Ltd., http://www.symbian.com/developer
Richard Harrison, Symbian OS C++ for Mobile Phones (Volume 2), Wiley, 2003
16. Evaluate This Document

In order to improve the quality of documentation, we kindly ask you to fill in the document survey.